ARMY RESEARCH LABORATORY

# Software Change-Merging in Dynamic Evolution

CPT David A. Dampier
U.S. ARMY RESEARCH LABORATORY

Valdis Berzins
NAVAL POSTGRADUATE SCHOOL

ARL-TR-841 August 1995

## NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | August 1995 | Final, Aug-Sep 94 |

**4. TITLE AND SUBTITLE**

Software Change-Merging in Dynamic Evolution

**5. FUNDING NUMBERS**

N/A

**6. AUTHOR(S)**

David A. Dampier and Valdis Berzins

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

U.S. Army Research Laboratory
ATTN: AMSRL-SC-IS
115 O'Keefe Building, GIT
Atlanta, GA 30332-0800

Computer Science Department
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943

**8. PERFORMING ORGANIZATION REPORT NUMBER**

ARL-TR-841

**9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)**

**10.SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

This paper was presented at the 1994 Monterey Workshop on Increasing the Impact of Formal Methods in Software Engineering, September 1994.

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution is unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

This position paper outlines a formal method for applying change-merging tools in dynamic evolution. During software evolution, different variations of a software system are generally developed. The need to apply a common change to each of these different versions will likely occur during the lifetime of the system. It may also be desirable to combine the unique capabilities of two different versions into a new version. Because these software systems can be very large, tools that automatically perform these tasks are desirable. Change-merging provides the capability for such a tool.

**14. SUBJECT TERMS**

change-merging, formal methods, software evolution, prototyping, automated maintenance

**15. NUMBER OF PAGES**

11

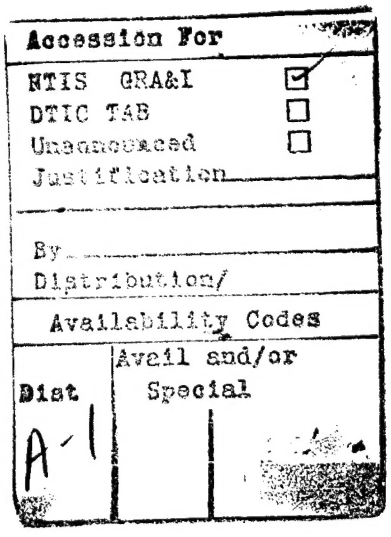
**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS





INTENTIONALLY LEFT BLANK.

## 1. INTRODUCTION

During software evolution, different variations of a software system are generally developed. The need to apply a common change to each of these different versions will likely occur during the lifetime of the system. It may also be desirable to combine the unique capabilities of two different versions into a new version. Because these software systems can be very large, tools that automatically perform these tasks are desirable. Change-merging provides the capability for such a tool.

## 2. EVOLUTIONARY PROTOTYPING

Rapid prototyping is an evolutionary approach to software development that was introduced to overcome the following weaknesses of traditional approaches:

(1) fully developed software systems that do not satisfy the customer's needs, or are obsolete upon release

(2) no capability for accurately evaluating real-time requirements before the software system has been built

Rapid prototyping overcomes these weaknesses by increasing customer interaction during the requirements engineering phase of development, providing executable specifications that can be evaluated for conformance to real-time requirements, and producing a production software system in a fraction of the time required using traditional methods. Rapid prototyping allows the user to get a better understanding of requirements early in the conceptual design phase of development. It involves the use of software tools to rapidly create concrete executable models of selected aspects of a proposed system to allow the user to view the model and make comments early. The prototype is rapidly reworked and redemonstrated to the user over several iterations until the designer and the user have a precise view of what the system should do. In this approach to rapid prototyping, software systems can be delivered incrementally as parts of the system become fully operational (Dampier 1994).

## 3. EVOLUTION IN CAPS

The Computer-Aided Prototyping System (CAPS) is an evolutionary prototyping system designed to prototype embedded, real-time systems (Luqi and Ketabchi 1988). CAPS consists of a set of prototyping

tools connected together by a graphical user interface. One of these tools is an Evolution Control System that not only provides version and configuration control for the software system, but also provides project management control in the form of scheduling development tasks and automatic assignment of designers to those tasks. In the version and configuration control model for the system, development histories are represented using variations and versions. Each variation number represents a parallel development history, and the version number represents the number of different versions in that particular variation. A variation/version number of 3.5 for a prototype means that this is the fifth version in the third variation.

## 4. CHANGE-MERGING

Change-merging is an integral part of the evolution methodology. During evolutionary development, multiple variations of a large system are likely to be developed. This can happen when independent development teams are working on different aspects of a system, or when alternate possible solutions to a problem are explored in different ways. Change-merging will allow the combination of these independently developed variations to be done automatically, ensuring that the resultant system is semantically correct, with respect to all of the input variations, or it will report all conflicts preventing correct change-merging. This technology encourages the designer to explore multiple solutions to a problem, and to spread the development workload in a large project without concern for the subsequent integration of these independent efforts (Dampier 1994).

Change-merging is a process by which significant changes between a base version of a software system and multiple modified versions can be isolated and combined into a single program as shown in Figure 1. As long as the changes do not conflict with one another, the result will be a program with the capabilities of all of the modified versions. Syntax-based change-merging methods like the revision control system (RCS) and source code control system (SCCS) do this by manipulating code and can produce a result that is syntactically correct (Silverberg 1992; Tichy 1982). They cannot provide any guarantee of correctness, however, so semantics-based methods are needed.

## 5. CHANGE-MERGING IN EVOLUTION

Software change-merging can be used in several different ways in software evolution. As we already stated, it can be used to combine different changes to the same base program. It can also provide a way to update multiple existing versions of a program with a change made to the common base version as

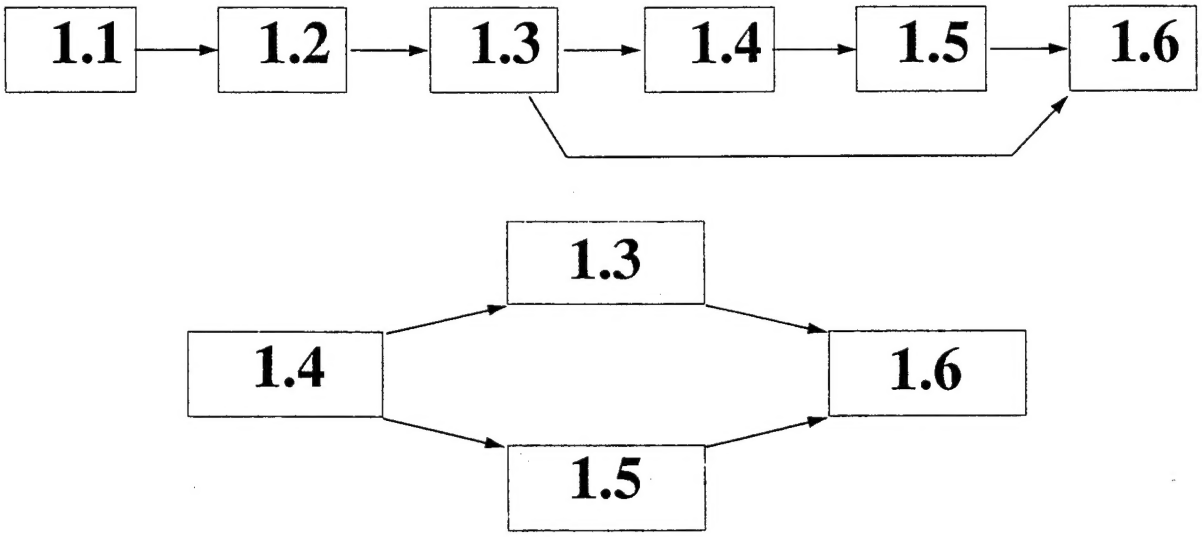



Figure 1. Change-merging two modified versions of a common base version.

illustrated in Figure 2. In this example, version 1.1 is the base, versions 1.2 and 2.2 are the modified versions, and version 3.2 is the changed base. The result of each of these operations is a modified version updated with the common change. It can also be used to check consistency between independently developed versions. If a change-merge operation applied to two independently developed versions does not produce a conflict, then the versions are consistent.

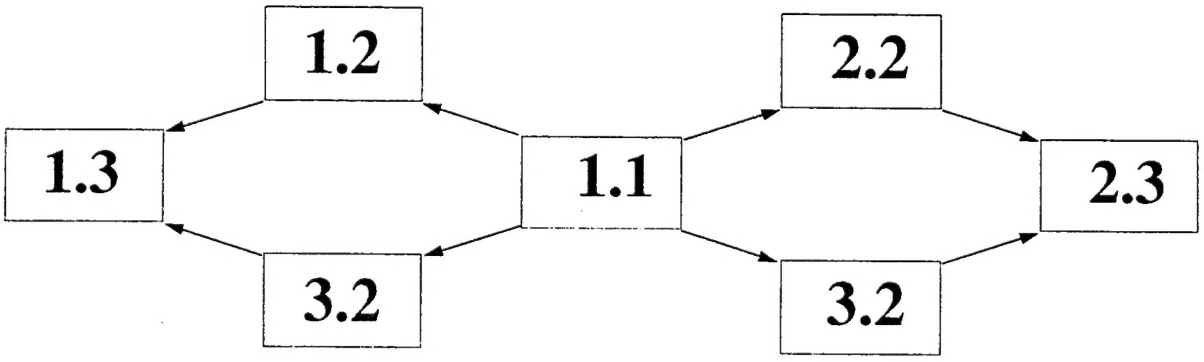



Figure 2. Updating multiple modifications with a change to the common base.

Another possible use of this technology is retracting changes from an evolution history. This idea is useful if after several iterations of the evolutionary process, the customer decides a feature of the software is no longer desired. Using change-merging, it should be possible to automatically retract the change as

long as the retraction does not cause a conflict in subsequent changes. The result of this operation would be a version that contains all of the capability in the most recent version of the system, except that contained in the retracted change, as shown in Figure 3.

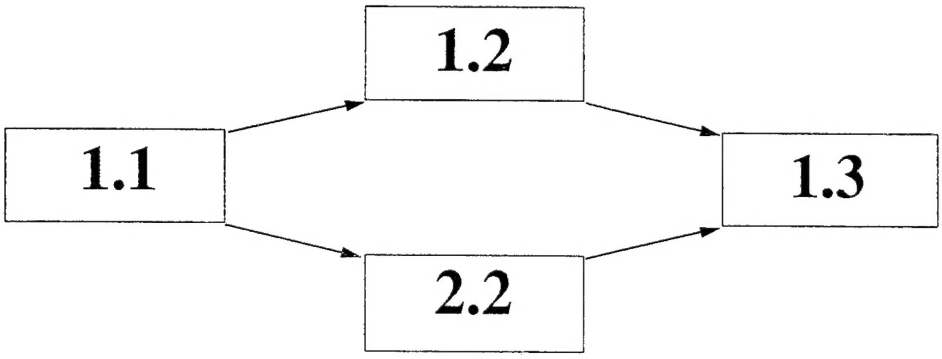



Figure 3. Retracting an earlier change from a subsequent version.

This example is designed to illustrate the removal of the change resulting in version 1.4 from version 1.5. Since 1.4 is the base version of the change-merge operation, the significant change from 1.4 to 1.3 is the retraction needed. This retraction must be preserved in the change-merged version 1.6.

## 6. SUMMARY AND FUTURE WORK

We have developed a slicing method for change-merging prototypes written in the prototype system description language (PSDL), the prototyping language associated with CAPS (Dampier 1994). This method will always produce a correct change-merged version if a conflict is not detected. Future work will include improving the resolution of the tool to prevent conflict reporting when no conflict exists, and trying to develop a change-merge method for other languages, perhaps Ada.

## 7. REFERENCES


Dampier, D. "A Formal Method for Semantics-Based Change-Merging of Software Prototypes." Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, June 1994.

Luqi, and M. Ketabchi. "A Computer Aided Prototyping System." IEEE Software, pp. 66–72, March 1988.

Silverberg, I. Source File Management with SCCS, Prentice Hall, Englewood Cliffs, NJ, 1992.

Tichy, W. "Design, Implementation, and Evaluation of a Revision Control System." Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, pp. 58–67, September 1982.

INTENTIONALLY LEFT BLANK.

## BIBLIOGRAPHY

Badr, S. "A Model and Algorithms for a Software Evolution Control System." Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, December 1993.

Berzins, V. "On Merging Software Extensions." Acta Informatica, Springer-Verlag, pp. 607–619, 1986.

Luqi, V. Berzins, and R. Yeh. "A Prototyping Language for Real Time Software." IEEE Transactions on Software Engineering, pp. 1409–1423, October 1988.

INTENTIONALLY LEFT BLANK.

| NO. OF COPIES | ORGANIZATION |
|---|---|
| 2 | ADMINISTRATOR<br>ATTN DTIC DDA<br>DEFENSE TECHNICAL INFO CTR<br>CAMERON STATION<br>ALEXANDRIA VA 22304-6145 |
| 1 | DIRECTOR<br>ATTN AMSRL OP SD TA<br>US ARMY RESEARCH LAB<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1145 |
| 3 | DIRECTOR<br>ATTN AMSRL OP SD TL<br>US ARMY RESEARCH LAB<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1145 |
| 1 | DIRECTOR<br>ATTN AMSRL OP SD TP<br>US ARMY RESEARCH LAB<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1145 |
| | ABERDEEN PROVING GROUND |
| 5 | DIR USARL<br>ATTN AMSRL OP AP L (305) |

| NO. OF COPIES | ORGANIZATION |
|---|---|
| 2 | NAVAL POSTGRADUATE SCHOOL<br>ATTN DR VARDIS BERZINS<br>COMPUTER SCIENCE DEPT<br>MONTEREY CA 93943 |
| 1 | DIR USARL<br>ATTN AMSRL SC I<br>115 OKEEFE BLDG<br>ATLANTA GA 30332-0800 |
| 50 | DIR USARL<br>ATTN AMSRL SC IS<br>115 OKEEFE BLDG<br>ATLANTA GA 30332-0800 |
| 2 | DIR USARL<br>ATTN AMSRL SC IS<br>CPT DAVID A DAMPIER<br>115 OKEEFE BLDG<br>ATLANTA GA 30332-0800 |
| | ABERDEEN PROVING GROUND, MD |
| 1 | DIR USARL<br>ATTN AMSRL SC |

## USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-TR-841 Date of Report August 1995

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

_____

_____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.)

_____

_____

_____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

_____

_____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

_____

_____

_____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)

**(DO NOT STAPLE)**